Clipper

จเร เลิศสุดวิชัย

เทคนิคการ เพิ่มความเร็ว ให้คลิปเปอร์ ตอนที่ 2

หลากหลายเทคนิค ที่น่าสนใจในการ เขียนโปรแกรมคลิปเปอร์ ให้ทำงานได้เร็วขึ้น

ในตอนที่แล้วได้เสนอวิธีการเพิ่มความเร็วให้โปรแกรมคลิปเปอร์ในบางส่วนไปบ้างแล้ว ในฉบับนี้ก็จะเสนอเทคนิคส่วนอื่นต่อไปดังนี้

ใช้นิพจน์ขยาย (Extended Expressions) แทนการใช้มาโคร

การทำงานของมาโครในคลิปเปอร์นั้นมีขั้นตอนมากมาย เริ่มตั้งแต่การแทนค่าตัวแปรมาโคร จากนั้นต้องทำการแปลคำสั่งที่ได้จากการแทนค่าในขั้นต้นแล้วจึงจะเริ่มทำงาน จะเห็นว่าโปรแกรมที่ใช้มาโครจะมีการทำงานที่ช้ากว่าโปรแกรมทั่วไป ดังนั้นเพื่อให้โปรแกรมที่พัฒนามีการทำงานที่เร็วขึ้นจึงควรลดการใช้มาโครในโปรแกรม ซึ่งในคลิปเปอร์ตั้งแต่รุ่น 5.0 ก็ได้มีรูปแบบคำสั่งพิเศษที่เรียกว่านิพจน์ขยายสำหรับใช้แทนมาโคร นิพจน์ขยายนี้จะถูกใช้ในขั้นของการแปลโปรแกรมทันที ซึ่งจะทำให้ไม่เสียเวลาแปลอีกขณะโปรแกรมทำงาน และโปรแกรมก็จะทำงานได้เร็วขึ้น

รูปแบบของนิพจน์ขยายนั้นก็เพียงนำเอานิพจน์ที่ต้องการใส่ไว้ในวงเล็บเท่านั้น ตัวอย่างเช่น คำสั่ง SET COLOR ซึ่งระบุสีเป็นข้อความ โดยหากใช้ตัวแปรมาโครคำสั่งเขียนได้ดังนี้

```
set color to & blue
```

แต่สามารถจะแทนได้ด้วยการใช้นิพจน์ขยายได้

```
set color to (blue)
```

ซึ่งความจริงก็สามารถใช้ฟังก์ชัน setcolor แทนได้ดังนี้

```
setcolor (blue)
```

มีหลายคำสั่งสามารถจะเปลี่ยนจากมาโครเป็นนิพจน์ขยายได้ทันที เช่น คำสั่งการเปิดไฟล์ข้อมูล

```
use &filename. index &ntxname.
```

สามารถเปลี่ยนเป็นนิพจน์ขยายได้ดังนี้

```
use (filename) index (ntxname)
```

ใช้โค้ดบล็อก (Code Block) แทนมาโคร

โค้ดบล็อกเป็นแบบข้อมูลชนิดพิเศษที่คลิปเปอร์รุ่น 5.0 เตรียมไว้สำหรับใช้งานแทนมาโครโดยเฉพาะ ข้อมูลแบบโค้ดบล็อกนี้จะเป็นชุดคำสั่งภาษาเครื่องซึ่งได้รับการแปลแล้วตั้งแต่ขั้นตอนการคอมพายล์โปรแกรม และพร้อมที่จะทำงานทันที ซึ่งจะคล้ายกับนิพจน์ขยาย แต่สามารถที่จะใช้ชุดคำสั่งที่ซับซ้อนได้และสามารถส่งผ่านพารา-มิเตอร์ได้ด้วย ตัวอย่างเช่น โปรแกรมที่เคยใช้มาโครดังในรูปที่ 1

จากโปรแกรมในรูปที่ 1 นี้ทุก ๆ รอบของการทำงานในบรรทัดของคำสั่ง IF จะมีการแทนค่าตัวแปรมาโคร แปลคำสั่งใหม่แล้วทำงาน ซึ่งเห็นว่าหากมีการทำงานหลาย ๆ รอบเวลาที่ถูกใช้ไปในขั้นของมาโครก็มากขึ้นด้วย เป็นผลให้การทำงานของโปรแกรมช้าลง

```
cMacro = "CUSTOMER → BirthDay < CTOD ("01/ 01/70")"
DO WHILE .NOT. EOF ( )
      IF &cMacro.
            .
            .
      ENDIF
      SKIP
ENDDO
```

รูปที่ 1 แสดงโปรแกรมที่เคยใช้มาโคร

```
cCodeBlock = ( || CUSTOMER → BirthDay < CTOD ("01/01/70"))
DO WHILE .NOT. EOF ( )
      IF EVAL (cCodeBlock)
            .
            .
      ENDIF
      SKIP
ENDDO
```

รูปที่ 2 แสดงโปรแกรมที่เปลี่ยนจากมาโครมาใช้โค้ดบล็อก

แต่ถ้าเปลี่ยนมาใช้โค้ดบล็อกจะทำให้โปรแกรมทำงานได้เร็วขึ้นกว่าเดิม ดังในรูปที่ 2

ใช้พรีโปรเซสเซอร์ (Preprocessor) แทนการใช้ตัวแปร

การเขียนโปรแกรมโดยทั่วไป ผู้เขียนมักต้องการให้โปรแกรมอ่านแล้วเข้าใจง่าย จึงมักนิยมใช้ชื่อสัญลักษณ์แทนในการอ้างถึงค่าคงที่ที่จะใช้งาน ซึ่งผู้ที่เคยเขียนโปรแกรมในคลิปเปอร์รุ่นเก่ามาก่อนจะพบว่าทางเดียวที่จะใช้ชื่อสัญลักษณ์ได้ก็ด้วยการตั้งเป็นตัวแปรแล้วใช้ชื่อให้สื่อความหมาย ซึ่งจากตอนที่แล้วได้กล่าวถึงเรื่องของการใช้ตัวแปรไปแล้วว่าเป็นส่วนหนึ่งที่ทำให้โปรแกรมทำงานช้า โดยเฉพาะตัวแปรแบบพายเวต

ตัวอย่างการตรวจสอบการกดแป้นพิมพ์ซึ่งใช้ฟังก์ชัน INKEY ( ) หรือ LASTKEY ( ) ในการตรวจสอบจะต้องเปรียบเทียบค่าที่ส่งออกมาจากฟังก์ชันว่าตรงกับค่าตัวเลขอะไร เช่น ต้องการตรวจสอบการกด Page Up จะเขียนได้ดังนี้

```
IF LASTKEY ( ) == 18
```

แต่หากต้องการให้โปรแกรมอ่านเข้าใจง่ายขึ้นส่วนใหญ่ผู้เขียนโปรแกรม

ก็จะเปลี่ยนเป็น

```
PAGE_UP = 18
IF LASTKEY ( ) = = PAGE_UP
```

ซึ่งหากเป็นคลิปเปอร์รุ่นเก่า ชื่อ PAGE_UP จะถูกสร้างเป็นตัวแปรพายเวตทันที และหากในโปรแกรมมีการใช้ในลักษณะนี้หลายจุดก็จะทำให้เกิดตัวแปรชนิดพายเวตขึ้นมากมาย ผลลัพธ์ก็คือโปรแกรมจะทำงานช้าลง

วิธีการใช้ชื่อแทนค่าข้อมูลเป็นสิ่งดี ดังนั้นในคลิปเปอร์รุ่นใหม่จึงได้เตรียมพรีโปรเซสเซอร์สำหรับใช้งานโดยเฉพาะ ซึ่งผู้เขียนโปรแกรมสามารถใช้พรีโปรเซสเซอร์กำหนดชื่อเพื่อใช้แทนค่าข้อมูลที่ต้องการได้ ชื่อเหล่านี้จะปรากฏอยู่เฉพาะในโปรแกรมฉบับ (Source Program) เท่านั้น แต่เมื่อโปรแกรมทำงานจะกลายเป็นค่าข้อมูลจริง

การประกาศชื่อแทนค่าข้อมูลจะต้องใช้คำประกาศ #define วางหน้าชื่อแล้วตามด้วยค่าข้อมูลที่จะถูกแทน ตัวอย่างเช่น

```
#define PAGE_UP      18
#define PAGE_DOWN    2
#define HOME         1
#define END          6
```

ในคลิปเปอร์ก็ได้นำพรีโปรเซสเซอร์มาใช้ในตัวอย่างดังในไฟล์ส่วนหัว (Header File) ในไดเรคทอรี INCLUDE เช่น ไฟล์ INKEY.CH ซึ่งเก็บการประกาศชื่อปุ่มกดของแป้นพิมพ์ที่ใช้แทนค่ารหัสคีย์

คอมพายล์โปรแกรมโดยระบุสวิตช์ /L

ขนาดของโปรแกรมเป็นส่วนหนึ่งที่มีผลโดยตรงต่อการทำงานของโปรแกรม โดยเฉพาะกับโปรแกรมที่เขียนด้วยคลิปเปอร์ซึ่งมีการจัดการหน่วยความจำเสมือนอยู่ในตัว นอกจากนั้นโปรแกรมขนาดเล็ก ๆ ยังทำให้การก๊อบปี้ไปให้ผู้ใช้หรือการเก็บรักษาทำได้สะดวกขึ้น

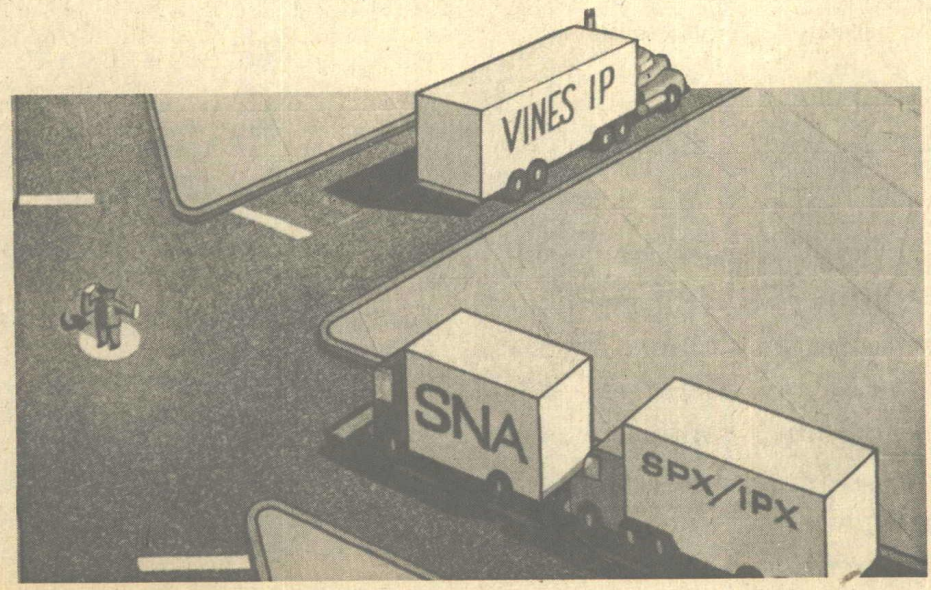
วิธีหนึ่งที่จะช่วยลดขนาดของโปรแกรมคลิปเปอร์คือการคอมพายล์โดยระบุสวิตช์ /L การระบุสวิตช์ตัวเลือกนี้จะให้คลิปเปอร์ตัดหมายเลขบรรทัดของโปรแกรมต้นฉบับออกจากออปเจ็กต์ไฟล์ ซึ่งทำให้โปรแกรมที่ได้มีขนาดเล็กลง แต่ตามปกติแล้วหมายเลขบรรทัดของโปรแกรมต้นฉบับที่มีอยู่ในโปรแกรมผลลัพธ์ของคลิปเปอร์จะมีประโยชน์มาก เพราะเมื่อโปรแกรมมีการทำงานผิดพลาด ก็จะมีการรายงานบอกถึงสาเหตุที่ทำให้โปรแกรมทำงานผิด พร้อมทั้งระบุหมายเลขบรรทัดคำสั่งในไฟล์โมดูลต้นฉบับที่ทำให้เกิดปัญหาด้วย ซึ่งเป็นข้อมูลที่คลิปเปอร์จัดเก็บเอาไว้ในขั้นตอนของการคอมพายล์นั่นเอง ตัวอย่างเช่น การรายงานข้อผิดพลาดดังนี้

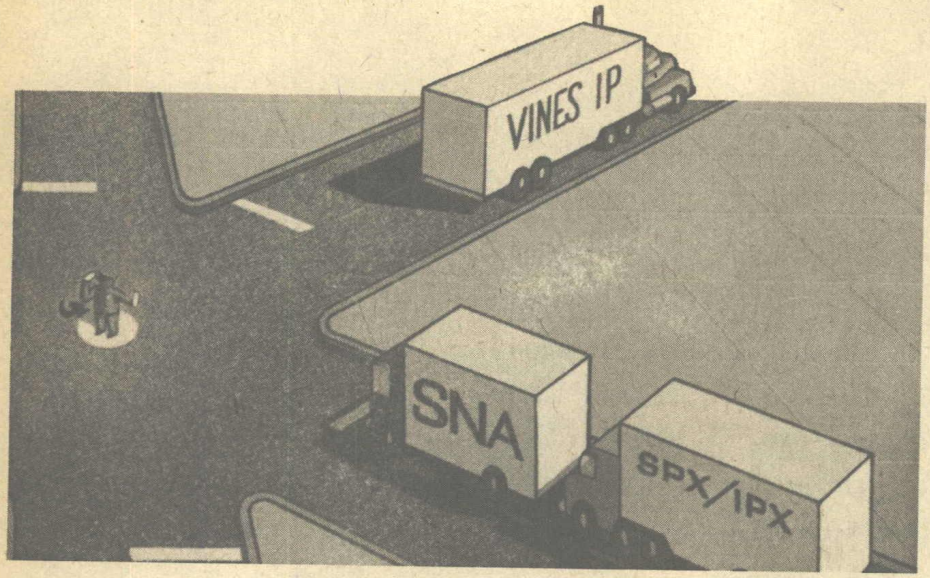
```
Error BASE/1097 Argument error: LEN
Called from Main (5)
```

หมายเลขบรรทัดนี้ทำให้ผู้เขียนโปรแกรมสามารถแก้ไขโปรแกรมได้ถูกจุด แต่หากระบุเลือกการคอมพายล์ด้วยสวิตช์ /L และเมื่อเกิดความผิดพลาดของโปรแกรมขึ้นจะมีการรายงานเพียงสาเหตุเท่านั้น ส่วนหมายเลขบรรทัดจะกลายเป็นศูนย์ เช่น ตัวอย่างการรายงานผลดังนี้

```
Error BASE/1097 Argument error: LEN
Called from Main (0)
```

ดังนั้นผู้เขียนโปรแกรมจะต้องเลือกเอาระหว่างขนาดของโปรแกรมกับข้อมูลสำหรับการดีบักโปรแกรม อย่างไรก็ตาม ผู้เขียนแนะนำว่าหากโปรแกรมที่พัฒนาสามารถทำงานโดยไม่มีข้อผิดพลาดแล้วควรจะระบุเลือกการคอมพายล์ด้วยสวิตช์ /L





คอมพายล์โปรแกรมโดยระบุสวิตช์ /N

การเขียนโปรแกรมในคลิปเปอร์เวลาคอมพายล์โปรแกรมโดยไม่ได้ระบุสวิตช์ /N คลิปเปอร์จะสร้างฟังก์ชันที่มีชื่อตามชื่อไฟล์โมดูลต้นฉบับ ซึ่งเป็นคุณสมบัติที่ต้องติดมาจากคลิปเปอร์รุ่นซัมเมอร์ 87 การสร้างฟังก์ชันจะเกิดขึ้นทุกครั้งแม้ว่าจะมีการประกาศชื่อฟังก์ชันในไฟล์แล้วก็ตาม

ลักษณะดังกล่าวจะทำให้เกิดข้อเสียหลายประการ เรื่องแรกก็คือการเสียพื้นที่ของโปรแกรมในการเก็บชื่อฟังก์ชันที่ถูกสร้างขึ้นมาโดยที่ไม่ได้ใช้งานเลย ประการที่สองคือจะทำให้เกิดชื่อฟังก์ชันที่เหมือนกันสองชื่อ หากในไฟล์โปรแกรมต้นฉบับนั้นได้ตั้งชื่อฟังก์ชันเป็นชื่อเดียวกับชื่อไฟล์โปรแกรมต้นฉบับ ซึ่งถ้าคลิปเปอร์ตรวจสอบพบในขณะคอมพายล์ก็จะรายงานทันที ซึ่งวิธีแก้ก็คือต้องไปเปลี่ยนชื่อฟังก์ชันไม่ให้ตรงกับชื่อไฟล์

ผลเสียประการสุดท้ายก็คือ จะทำให้ไม่สามารถประกาศตัวแปรแบบสแตติกภายนอกได้เลย ตัวแปรแบบสแตติกแบบนี้จำเป็นต้องใช้ในการจำกัดขอบเขตการใช้งาน

ใช้การคำนวณค่าลอจิกแบบรวบรัด (Short Circuit)

วิธีการง่าย ๆ ในการเพิ่มความเร็วให้โปรแกรมคลิปเปอร์อีกวิธีหนึ่งก็คือการวางตำแหน่งของนิพจน์ลอจิกในคำสั่งควบคุมการทำงานของโปรแกรม (Control Flow) เพราะตั้งแต่คลิปเปอร์รุ่น 5.0 เป็นต้นไปได้เพิ่มเทคนิคการคำนวณค่าลอจิกแบบรวบรัด ทำให้สามารถทราบผลการคำนวณได้อย่างรวดเร็วโดยไม่ต้องคำนวณหาค่าทั้งนิพจน์ ดังตัวอย่างโปรแกรมนี้

```
x := .F.
IF x .AND. LongExpression ( )

.

ENDIF
```

จากโปรแกรมตัวอย่าง เมื่อคลิปเปอร์ทำงานจนถึงคำสั่ง IF ที่ต้องมีการตัดสินใจเลือกการทำงานคำสั่งต่อไป ซึ่งการเลือกนี้จะใช้ผลการคำนวณค่าลอจิกของนิพจน์ การคำนวณค่าลอจิกแบบรวบรัดของคลิปเปอร์จะเริ่มจากทางด้านซ้ายมือของนิพจน์ไปทางขวา โดยหากคลิปเปอร์คาดเดาผลลัพธ์ของนิพจน์ได้ก่อนก็จะหยุดการคำนวณทันที ดังในตัวอย่างโปรแกรมนี้คลิปเปอร์จะทราบผลของนิพจน์นี้ทันทีที่ว่าผลลัพธ์คือเท็จ เมื่อคำนวณถึงตัวแปร x และเครื่องหมาย .AND. โดยที่คลิปเปอร์จะไม่เข้าไปทำงาน LongExpression

สิ่งนี้คือความสามารถพิเศษที่คลิปเปอร์เพิ่มมาให้ แต่ก็เป็นหน้าที่ของผู้เขียนโปรแกรมด้วยในการที่จะจัดวางตำแหน่งของนิพจน์ โดยให้เลือกเอานิพจน์สั้น ๆ หรือนิพจน์ที่มีการคำนวณง่ายมาไว้ด้านหน้าของเครื่องหมาย .AND และ .OR